

## Formale Systeme, WS 2009/2010

### Praxisaufgabe 2: Theorembeweiser KeY

Abgabe am 14.2.2010.

Für die vollständige Lösung dieser Praxisaufgabe – sie besteht aus den beiden Teilaufgaben 1 und 2 – erhalten Sie 1,5 Bonuspunkte für die Abschlussklausur (bitte beachten Sie die Erläuterung zu Bonuspunkten auf der Webseite zur Vorlesung).

## A Informationen zum KeY-System

### A.1 Allgemeines

**Was ist KeY?** Zusammen mit unseren Partnern an der Chalmers University of Technology in Göteborg wird an unserem Institut das KeY-System entwickelt. Es ist ein Softwareverifikationswerkzeug, mit dem die Übereinstimmung von Java Card-Software und ihrer Spezifikation formal bewiesen werden kann.

Die Logik, auf der das KeY-System basiert, ist eine dynamische Prädikatenlogik. In dieser dynamischen Logik ist die in der Vorlesung eingeführte Prädikatenlogik vollständig enthalten. Deshalb können wir KeY auch benutzen, um rein prädikatenlogische Probleme zu formulieren und zu beweisen.

**Literatur zu KeY** Auf der Internetseite der Vorlesung steht eine Einführung zu Beweisen von prädikatenlogischen Formeln mit KeY ([KeYIntro.pdf](#)). Bitte arbeiten Sie diese Einführung durch.

Für weitergehende Fragen bietet sich die Lektüre des KeY-Buches [BHS07] an und darin vor allem Kapitel 10, das eine tiefere Einführung in KeY bietet. Kapitel 2 des Buches erklärt fundiert die prädikatenlogischen Grundlagen, auf denen KeY basiert.

**Installation** Das KeY-System besitzt eine graphische Benutzeroberfläche und ist komplett in Java geschrieben, so daß es auf jeder Plattform, für die eine virtuelle Maschine für Java zur Verfügung steht, lauffähig ist.

Version 1.4 ist die zuletzt veröffentlichte Version und kann über die KeY-Homepage<sup>1</sup> bezogen werden. Wenn Sie die Software „Java Web Start“ installiert haben (auf fast allen modernen Systemen mit Java der Fall), können Sie KeY auch direkt aus dem Internetbrowser heraus starten, ohne etwas installieren zu müssen.

### A.2 Hinweis zur Konfiguration von KeY

Die automatische Beweisbarkeit hängt jeweils von der Problemformulierung sowie von den verwendeten Beweisereinstellungen ab.

- Für Teilaufgabe 1 empfiehlt es sich, im Bereich unten links in KeY auf der Registerkarte „Proof Search Strategy“ die Einstellung **FOL** zu wählen.
- Für Teilaufgabe 2 wählen Sie am besten **Java DL** und belassen alle weiteren Schalter in ihrer Standardstellung.

---

<sup>1</sup><http://www.key-project.org/download/key.html>

### A.3 KeY-Problemdateien zu den Teilaufgaben

- Für Teilaufgabe 1 erstellen Sie bitte selbst eine KeY-Problemdatei.
- Für Teilaufgabe 2 finden Sie auf der Webseite der Vorlesung die Datei `inssort.key` zum Herunterladen vor. Diese müssen Sie vervollständigen (wie unten im Abschnitt zu Teilaufgabe 2 beschrieben).

### A.4 Abgabe der Lösungen

Bitte speichern Sie die beiden Beweise in KeY als „key.proof“-Dateien ab. Auf der Webseite der Vorlesung steht Ihnen ein Formular zur Verfügung, um die beiden Lösungen einzureichen.

Für unvollständige Beweise werden Punkte auch anteilig vergeben.

## B Teilaufgabe 1: Der Fall „Tante Agathe“ wird aufgerollt

Eine bereits bekannte Aufgabe, die noch auf ihre Lösung wartet, zu Beginn: Auf dem 4. Übungsblatt wurde in Aufgabe 4 ein logisches Puzzle vorgestellt, das Teil einer großen Benchmarkbibliothek für Beweiser ist. Über der dort eingeführten Signatur lassen sich die gegebenen Indizien folgendermaßen formalisieren:<sup>2</sup>

$$\begin{aligned} & \forall x(x \doteq a \vee x \doteq b \vee x \doteq c) \\ \wedge & \exists x(kills(x, a)) \\ \wedge & \forall x \forall y (kills(x, y) \rightarrow hates(x, y)) \\ \wedge & \forall x (hates(a, x) \rightarrow \neg hates(c, x)) \\ \wedge & \forall x \forall y (kills(x, y) \rightarrow \neg richer(x, y)) \\ \wedge & \forall x ((\neg richer(x, a) \vee hates(a, x)) \rightarrow hates(b, x)) \\ \wedge & \forall x \exists y \neg hates(x, y) \\ \wedge & \forall x (\neg x \doteq b \rightarrow hates(a, x)) \\ \wedge & \neg a \doteq b \end{aligned}$$

- Formalisieren Sie nun auch noch die Aussage „Tante Agathe hat sich selbst umgebracht.“,
- schreiben Sie eine Problembeschreibungsdatei für den Beweiser KeY und
- beweisen Sie mit KeY, daß aus den Indizien folgt, daß Tante Agathe sich selbst umgebracht haben muß.

## C Teilaufgabe 2: Abstrakte Datentypen

**Aufgabe.** In dieser Aufgabe betrachten wir den termerzeugten Datentyp „Liste von Zahlen“. Ziel ist es, eine Funktion zu definieren, die die Liste sortiert (mittels Insertion-Sort), und formal ihre Korrektheit zu beweisen. Dies ist eine typische Fragestellung der *Programmverifikation*. Die Definition der Sortieroperation in der Logik entspricht dabei einer Implementierung in einer funktionalen Programmiersprache.

---

<sup>2</sup>Die letzten beiden Aussagen waren auf dem Übungsblatt nicht enthalten, sind aber notwendig, um die Behauptung beweisen zu können

**Signatur.** Wir betrachten zwei Sorten von Termen bzw. Elementen: ganze Zahlen (Sorte `int`) und Listen von ganzen Zahlen (Sorte `lst`). In der Signatur gibt es u.a.:

die Konstruktoren

`nil` :  $\rightarrow \text{lst}$   
`cons` :  $\text{int} \times \text{lst} \rightarrow \text{lst}$

die Mutatoren

`insert` :  $\text{int} \times \text{lst} \rightarrow \text{lst}$  (sortiert die Zahl richtig ein in die Liste)  
`sort` :  $\text{lst} \rightarrow \text{lst}$  (sortiert die Liste)

den Inspektor

`count` :  $\text{int} \times \text{lst} \rightarrow \text{int}$  (wie oft kommt die Zahl vor in der Liste?)

die Prädikate

`sorted` :  $\text{lst} \rightarrow \text{bool}$  (ist die Liste sortiert?)  
`least` :  $\text{int} \times \text{lst} \rightarrow \text{bool}$  (es gibt keine kleinere Zahl in der Liste)

Wir beschränken uns nur auf „kanonische“ Interpretationen, in denen die Sorte `int` der Menge  $\mathbb{Z}$  der ganzen Zahlen und `lst` der Menge aller (endlichen) Listen von ganzen Zahlen entspricht. Die Menge der Listen wird von den beiden Funktionen `nil` und `cons` erzeugt. Die Signatur beinhaltet außerdem die üblichen Funktionen und Prädikate auf `int`, die ebenfalls kanonisch interpretiert werden. Die Variablen  $n$  und  $m$  seien im Folgenden stets vom Typ `int` und  $l$  vom Typ `lst`.

**Induktionsregel.** Da wir nur bestimmte Interpretationen betrachten, können wir den in der Vorlesung vorgestellten Sequenzenkalkül um weitere Regelschemata erweitern. Um Aussagen über Listen beweisen zu können, nehmen wir das Regelschema für die strukturelle Induktion über Listen hinzu:

$$\frac{\begin{array}{l} \Gamma \Rightarrow \{l/\text{nil}\}\varphi, \Delta \\ \Gamma \Rightarrow \forall n \forall l (\varphi \rightarrow \{l/\text{cons}(n,l)\}\varphi), \Delta \end{array}}{\Gamma \Rightarrow \forall l \varphi, \Delta} \quad (\text{listInduction})$$

Diese Regel ist korrekt, da die Sorte `lst` von den beiden Konstruktoren `nil` und `cons` termerzeugt wird, also jedes Element der Sorte als Term über diesen Funktionen dargestellt werden kann.

**Datentypaxiome.** Die Funktionen `insert`, `sort`, `count` und die Prädikate `sorted`, `least` sind durch folgende Axiome (partiell) festgelegt:

$$\begin{array}{lll} \forall n \text{ least}(n, \text{nil}) & & (\text{leastNil}) \\ \forall n \forall m \forall l (\text{least}(n, \text{cons}(m, l)) \leftrightarrow n \leq m \wedge \text{least}(n, l)) & & (\text{leastCons}) \\ \\ \text{sorted}(\text{nil}) & & (\text{sortedNil}) \\ \forall n \forall l (\text{sorted}(\text{cons}(n, l)) \leftrightarrow \text{least}(n, l) \wedge \text{sorted}(l)) & & (\text{sortedCons}) \\ \\ \forall n \text{ count}(n, \text{nil}) & \doteq & 0 & (\text{countNil}) \\ \forall n \forall m \forall l \text{ count}(n, \text{cons}(m, l)) & \doteq & \text{count}(n, l)^3 + & (\text{countCons}) \\ & & (\text{if } n \doteq m \text{ then } 1 \text{ else } 0)^4 & \\ \\ \forall n \text{ insert}(n, \text{nil}) & \doteq & \text{cons}(n, \text{nil}) & (\text{insertNil}) \\ \forall n \forall m \forall l \text{ insert}(n, \text{cons}(m, l)) & \doteq & \dots & (\text{insertCons}) \\ \\ \text{sort}(\text{nil}) & \doteq & \text{nil} & (\text{sortNil}) \\ \forall n \forall l \text{ sort}(\text{cons}(n, l)) & \doteq & \dots \text{ (benutzen Sie insert)} & (\text{sortCons}) \end{array}$$

## C.1 Vervollständigung der Formalisierung des Datentyps

Auf der Seite zur Vorlesung finden Sie eine KeY-Datei, in der die obigen Axiome als Sequenzkalkülregeln formalisiert sind (die z.T. automatisch angewendet werden). Vervollständigen Sie die mit „...“ markierten Stellen.

## C.2 Die zu beweisende Aussage

Beweisen Sie mit KeY, daß die Funktion `sort` eine (beliebige) Liste von ganzen Zahlen korrekt sortiert:

$$\forall l (\text{sorted}(\text{sort}(l)) \wedge \forall n (\text{count}(n, \text{sort}(l)) \doteq \text{count}(n, l)))$$

Der erste Konjunkt sichert zu, daß `sort` ein sortiertes Ergebnis liefert. Der zweite stellt sicher, daß keine Elemente beim Sortieren der Liste verlorengehen oder dazukommen (daß also die sortierte Liste eine Permutation der Eingabe ist).

Hinweis: Versuchen Sie zunächst, die beiden Teile separat zu beweisen (die Teile sind unabhängig).

## C.3 Hilfreiche Lemmata

Mit einer einzigen Induktion kann die Aussage leider nicht bewiesen werden. Sie werden im Laufe des Beweises an Punkte kommen, an denen Lemmata benötigt werden, die selbst wieder durch Induktion zu beweisen sind. Benutzen Sie die (eingebaute) Regel `cut`, um ein Lemma einzufügen:

$$\frac{\overbrace{\Gamma, \varphi \Rightarrow \Delta}^{(A)} \quad \overbrace{\Gamma \Rightarrow \varphi, \Delta}^{(B)}}{\Gamma \Rightarrow \Delta} \text{ (cut)}$$

`cut` teilt den Beweis in zwei Äste auf:

- Ast A: Hier steht die eingefügte Formel  $\varphi$  links des Sequenzpfeiles. Damit steht sie auf diesem Ast als weitere Voraussetzung zur Verfügung und kann benutzt werden, um das ursprüngliche Ziel zu schließen (Verwenden des Lemmas).
- Ast B: Hier steht  $\varphi$  rechts des Sequenzpfeiles. Auf diesem Ast ist  $\varphi$  also das Beweisziel und muß gezeigt werden (Beweis des Lemmas).

Im folgenden sind einige hilfreiche Lemmata aufgelistet:

**Lemma 1**  $\forall l \forall n \forall m ((n \leq m \wedge \text{least}(m, l)) \rightarrow \text{least}(n, l))$

**Lemma 2**  $\forall l \forall n \forall m (\dots \rightarrow \text{least}(n, \text{insert}(m, l)))$

**Lemma 3** „insert erhält die Sortiertheit der Liste (bzgl. Prädikat `sorted`).“

**Lemma 4** soll das Zusammenspiel zwischen `count` und `insert` klären.

## Literatur

[BHS07] Bernhard Beckert, Reiner Hähnle, and Peter H. Schmitt, editors. *Verification of Object-Oriented Software: The KeY Approach*. LNCS 4334. Springer-Verlag, 2007.

<sup>3</sup>Korrigierte Version (in einer früheren Version des Aufgabenblattes stand hier `count(n, cons(m, l))` statt `count(n, l)`).

<sup>4</sup>`if  $\phi$  then  $t_1$  else  $t_2$`  ist ein sog. *bedingter Term* (engl. conditional term).

$$\text{val}_{D, I, \beta}(\text{if } \phi \text{ then } t_1 \text{ else } t_2) := \begin{cases} \text{val}_{D, I, \beta}(t_1) & \text{wenn } \text{val}_{D, I, \beta}(\phi) = W \\ \text{val}_{D, I, \beta}(t_2) & \text{wenn } \text{val}_{D, I, \beta}(\phi) = F \end{cases}$$